University of California Santa Cruz
Department of Computer Engineering
Lab Experiment Report # 2

Adder and LED Display

Author: Kyle Jeffrey
Lab Partner: NA
Due Date: 1/25/18

Objective

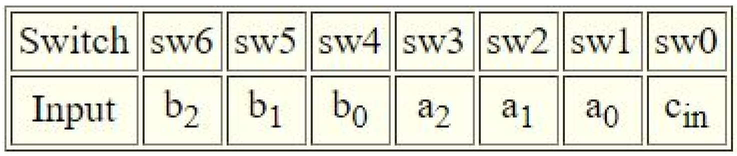
The purpose of this lab was to create a 3 bit adder using 3 rippled full adders in verilog by instantiating modules. The output of the adder then went through some logic to turn 3 inputs into 6 outputs that would display on the rightmost LED.

Kyle Jeffrey
Prof Schlag
CE 100
1/18/18

Part One: Adder Design:

Using the inputs below, designing a 3 bit adder required two steps:

- Make a module for a 1 bit adder
- Make a module using the 1 bit adder to create a 3 bit adder



Switch	sw6	sw5	sw4	sw3	sw2	sw1	sw0
Input	b_2	b_1	b_0	a_2	a_1	a_0	c_{in}

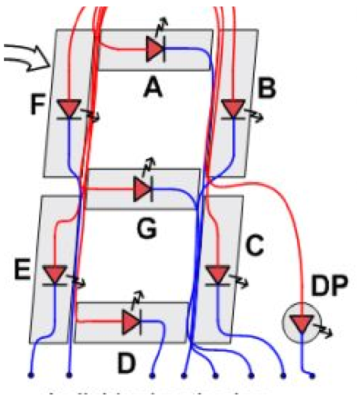
Method:

1 bit Adder: The logic for this was determined using SOP on the truth table of inputs a,b,cin and outputs cout, sum. The logic expressions simplified to sum = a ^ b ^ c, and cout = c (a ^ b) + ab.

3 bit Adder: Using the 1 bit adder, instantiate three instances and use wires connecting the carry out of the first adder to the carry in of the second, and the same for the second adder to the third. For the first adder, set carry in to zero to behave like a half adder. The final carry out is an overflow bit and wont be used.

Part Two: 7 Segment Display:

The 7 segment display has inputs CA - CG each powering a different segment on the LED display, requiring that I take a 4 bit vector (the sum of our switch inputs) and converting it to a 7-bit vector corresponding to CA through CG. The inputs an[3:0] control which of the 4 hex displays our segments are powering, and in this lab we only cared about an0.



Method:

Logic for 4-bit to 7-bit vector: The four bit number generates sixteen possible permutations that correspond to a number to be displayed on the LED. I began by writing out a truth table with the 4 inputs and 7 outputs and went number by number filling out the outputs. Each segment that needed to be lit up for that number had a 1 and each segment to be turned off had a 0. Using SOP, I had sixteen possible minterms to set the outputs equal to. In example, CA = m(0 , 2. 3, 5, 6, 7, 8, 9, 10, 12, 14, 15).

Images taken from Basys 3 Reference Manual

Corresponding to the segment display above, this means that these numbers have the top segment in them, which appears to be correct.

Part Three: Test Bench and Simulation:

Once downloading the given testbench file and correctly changing all of the i/o names for simulation, I began simulation debugging for ease of debug opposed to generating a bit stream everytime and trying to fix the bugs.

Method:

I used the simulation to make sure that:

1. The adder was adding correctly
2. The display logic printed the correct hex digit

Testing the adder: The adder was tested by setting the switches to at least eight different values. Once testing eight of the values, I had confidence that the logic was correct. The most important values were fifteen and sixteen which acted as boundary cases. If those both worked, then the middle values were also more likely to work. In practice, this was tested by setting sw[6:0] to different values and seeing what the wires s0, s1, and s2 equaled.

Testing the Hex Display: The hex display was less intuitive to debug. This was debugged at the same time as debugging the adder, but the logic for this was much easier to have a mistake on because of the number of equations. So, using the switches and going through values zero to sixteen was necessary in determining if the display was correct.

RESULTS:

This was the point at which all of the logic design was tested for error in the simulation. Upon initial debug the top segment (CA) was not displaying for zero and other digits and was fixed when realizing that an incorrect minterm was placed. Everything else appeared fine when debugging and it wasn't until loading the bitstream to the board that I noticed all of the digits were inverted. The logic for the hex's was done in SOP for active high and had to be inverted. Every other issue derived from syntax error in the Verilog code like just entering 1 instead of 1'b1.

Images taken from Basys 3 Reference Manual

Kyle Jeffrey
Prof Schlag
CE 100
1/18/18

Question 1: Document which pins of the FPGA you used and how they were connected to the switches and 7-segment displays

Answer: The connections were all discussed above, but to reiterate: the switches went into the adder and the sum from the adder was connected to the hex display which directly connected to the hex inputs.

Question 2: Determine the longest path from any input to any output in your 3-bit adder. (i.e. the highest number of gates that any input goes through before reaching an output.)

Answer: If the carry in is 1 and either input a or b is 1 but NOT both, then that bit will travel through and XOR gate, to an AND gate, and eventually pass through an OR gate, totaling 3 gates.

Question 3: For an *n*-bit adder determine the length of the longest path from any input to any output (in terms of *n*).

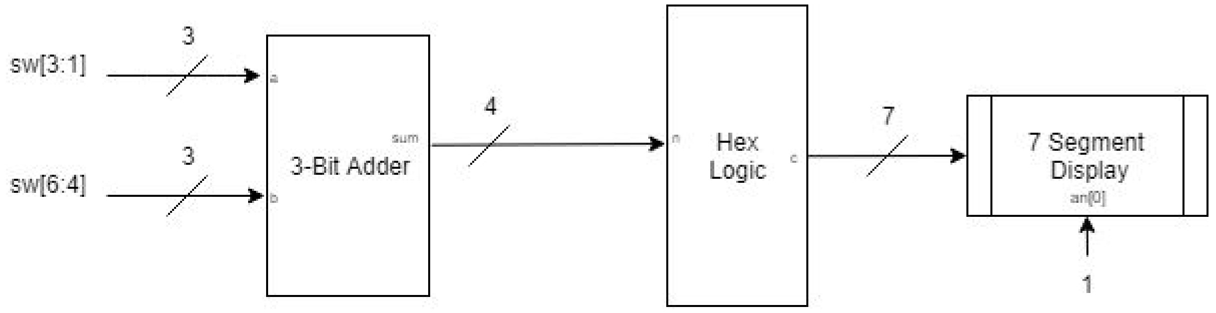
Answer: The longest path starts at the input a in the first adder and then travels out of the carry out port, back into the carry in port on the next adder, where the longest path is through 2 gates. Y = 3+ 2(n - 1)

Question 4: Calculate the number of possible input values (7-bit vectors) for your adder? Give the percentage of these that you tested in your simulation.

Answer: There were to 3-bit vectors added allowing for 16 possible values, but the last bit was aa carry in on the first adder. This bit only actually allowed 1 more value instead of considering a normal 2^n number of permutations. There were 17 possible sums and I simulated 15 out of 17.

Images taken from Basys 3 Reference Manual

Top Level Schematic



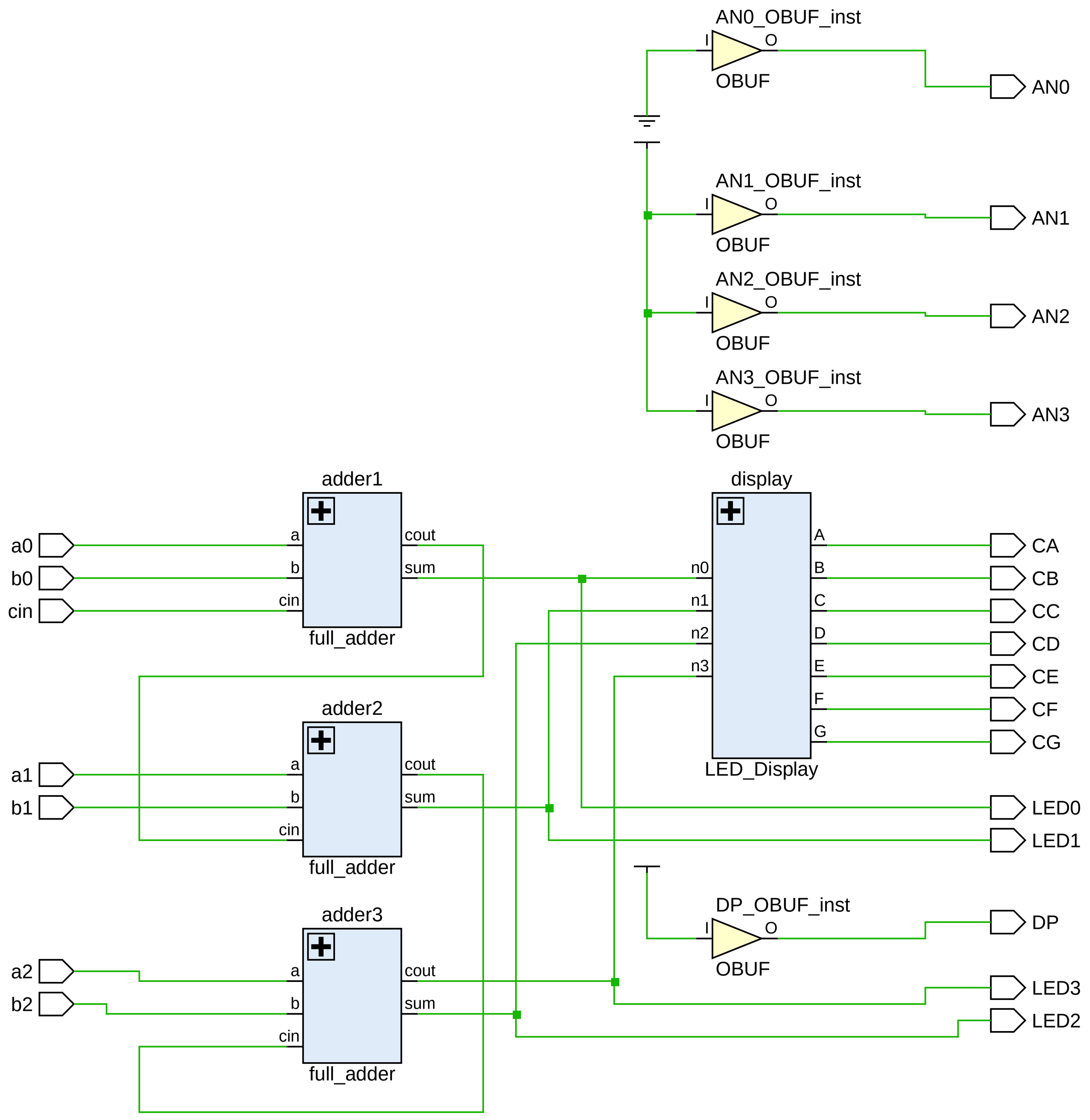
Conclusion

This lab made it very apparent how important the simulation test bench is for debugging in verilog. It was absolutely implausible to solely debug the program by regenerating the bitstream file every time a fix was added to the code. Upon doing the lab again, I would stress the importance of preparation and, though it seems obvious, neat handwriting. The longest period I spent on debugging dealt with reversing the indexes on the bus into the hex logic and just making dumb errors due to bad handwriting in my lab notebook. I could've optimised the adder section of the logic a bit by using the sign bit to determine whether to add 1 or 0.

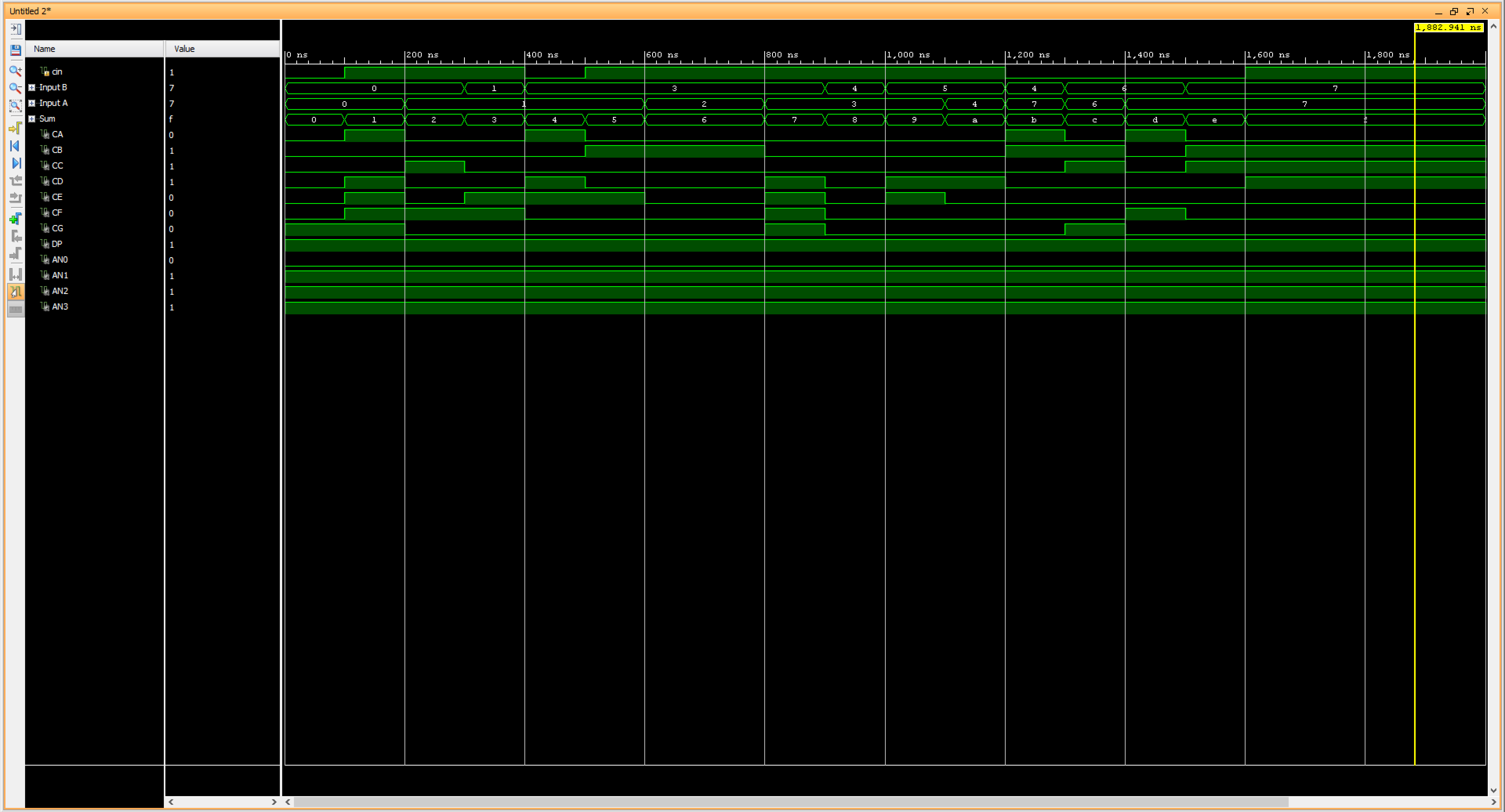
Images taken from Basys 3 Reference Manual

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 01/11/2019 03:59:20 PM
// Design Name: 
// Module Name: full_adder
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module full_adder(
    input a,
    input b,
    input cin,
    output sum,
    output cout
    );
    
    assign cout = cin & (a ^ b) | a & b;  // carryout bit is c(a xor b) or (a and b)
    assign sum = a ^ b ^ cin;              // sum bit is only 1 when there are odd
number of 1's
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/15/2019 10:42:21 AM
// Design Name:
// Module Name: LED_Display
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module LED_Display(

    input n3,
    input n2,
    input n1,
    input n0,
    output A,
    output B,
    output C,
    output D,
    output E,
    output F,
    output G
    );
    wire m0, m1, m2, m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15;

    assign m0 = ~n3 & ~n2 & ~n1 & ~n0;
    assign m1 = ~n3 & ~n2 & ~n1 & n0;
    assign m2 = ~n3 & ~n2 & n1 & ~n0;
    assign m3 = ~n3 & ~n2 & n1 & n0;
    assign m4 = ~n3 & n2 & ~n1 & ~n0;
    assign m5 = ~n3 & n2 & ~n1 & n0;
    assign m6 = ~n3 & n2 & n1 & ~n0;
    assign m7 = ~n3 & n2 & n1 & n0;
    assign m8 = n3 & ~n2 & ~n1 & ~n0;
    assign m9 = n3 & ~n2 & ~n1 & n0;
    assign m10 = n3 & ~n2 & n1 & ~n0;
    assign m11 = n3 & ~n2 & n1 & n0;
    assign m12 = n3 & n2 & ~n1 & ~n0;
    assign m13 = n3 & n2 & ~n1 & n0;
    assign m14 = n3 & n2 & n1 & ~n0;
    assign m15 = n3 & n2 & n1 & n0;


    assign A = ~(m0 | m2 | m3 | m5 | m6 | m7 | m8 | m9 | m10 | m12 | m14 | m15);
    assign B = ~(m0 | m1 | m2 | m3 | m4 | m7 | m8 | m9 | m10 | m13);
    assign C = ~(m0 | m1 | m3 | m4 | m5 | m6 | m7 | m8 | m9 | m10 | m11 | m13);
    assign D = ~(m0 | m2 | m3 | m5 | m6 | m8 | m11 | m12 | m13 | m14);
    assign E = ~(m0 | m2 | m6 | m8 | m10 | m11 | m12 | m13 | m14 | m15);
    assign F = ~(m0 | m4 | m5 | m6 | m8 | m9 | m10 | m11 | m12 | m14 | m15);
    assign G = ~(m2 | m3 | m4 | m5 | m6 | m8 | m9 | m10 | m11 | m13 | m14 | m15);


endmodule
```


AN0_OBUF_inst
I
O
OBUF
AN0
AN1_OBUF_inst
I
O
OBUF
AN1
AN2_OBUF_inst
I
O
OBUF
AN2
AN3_OBUF_inst
I
O
OBUF
AN3
adder1
a0
b0
cin
a
b
cin
cout
sum
full_adder
display
n0
n1
n2
n3
A
B
C
D
E
F
G
LED_Display
CA
CB
CC
CD
CE
CF
CG
adder2
a1
b1
a
b
cin
cout
sum
full_adder
LED0
LED1
DP_OBUF_inst
I
O
OBUF
DP
adder3
a2
b2
a
b
cin
cout
sum
full_adder
LED3
LED2

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/11/2019 04:07:30 PM
// Design Name:
// Module Name: top_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_adder(
    input a0,
    input a1,
    input a2,
    input b0,
    input b1,
    input b2,
    input cin,
    output DP,
    output CA,
    output CB,
    output CC,
    output CD,
    output CE,
    output CF,
    output CG,
    output AN0,
    output AN1,
    output AN2,
    output AN3,
    output LED3,
    output LED2,
    output LED1,
    output LED0
    );
    wire t, g, s3, s2, s1, s0;



    full_adder adder1(.a(a0), .b(b0), .sum(s0), .cout(t), .cin(cin));

    full_adder adder2(.a(a1), .b(b1), .sum(s1), .cout(g), .cin(t));

    full_adder adder3(.a(a2), .b(b2), .sum(s2), .cout(s3), .cin(g));



    //test
    LED_Display display(.n3(s3), .n2(s2), .n1(s1), .n0(s0), .A(CA), .B(CB), .C(CC),
.D(CD), .E(CE), .F(CF), .G(CG) );
    assign DP = 1;
    assign AN0 = 0;
    assign AN1 = 1;
    assign AN2 = 1;
    assign AN3 = 1;

    assign LED3 = s3;
    assign LED2 = s2;
    assign LED1 = s1;
    assign LED0 = s0;
endmodule
```



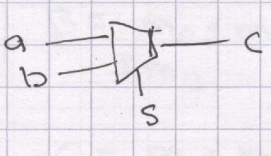
Name	Value
cin	1
Input B	7
Input A	7
Sum	f
CA	0
CB	1
CC	1
CD	1
CE	0
CF	0
CG	0
DP	1
AN0	0
AN1	1
AN2	1
AN3	1

```
Input B: 0 | 1 | 3 | 4 | 5 | 4 | 6 | 7
Input A: 0 | 1 | 2 | 3 | 4 | 7 | 6 | 7
Sum:     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f
```

Lab 2

$\bar{a}b + ab = b(\bar{a}+a)$
$= b(1)$
$= b$

Mux Review



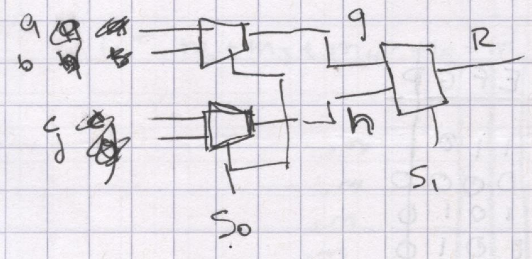
a	b	s	c
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$\bar{a}bs + a\bar{b}\bar{s} + ab\bar{s} + abs$

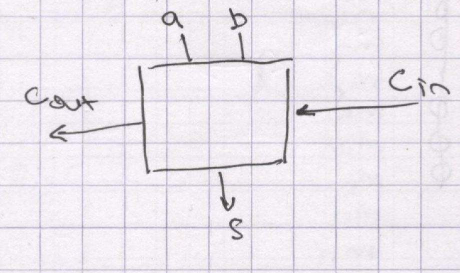
$s(\bar{a}b + ab) + \bar{s}(ab + a\bar{b})$

$\underline{sb + \bar{s}a}$ 2 to 1 mux

○ 4 to 1 mux



Full Adder



a	b	C_{in}	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

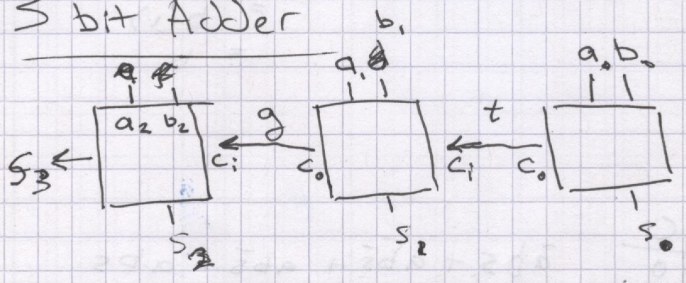
$b + \bar{b}c$

$C_o = \bar{a}bc_i + a\bar{b}c_{in} + ab\bar{c}_{in} + abc$
$= a(\bar{b}c + b\bar{c} + bc) + \bar{a}bc$
$= a(\bar{b}c + b) + \bar{a}bc$
$= a\bar{b}c + ab + \bar{a}bc$
$= c(a\bar{b} + \bar{a}b) + ab$
$= c(a \oplus b) + ab$

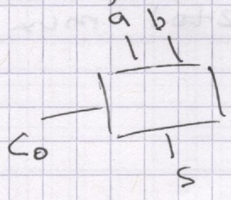
$S = a \oplus b \oplus c$

~~digit 1: [illegible]~~
~~Digit 2: b, d, f~~

3 bit Adder



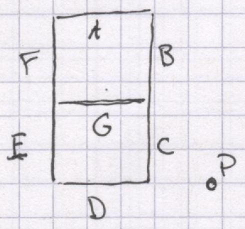
- Half Adder



a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	~~0~~ 1
1	1	1	0

$S = a \oplus b$
$C = ab$

7 Segment Display



n_3	n_2	n_1	n_0	A	B	C	D	E	F	G	P	
0	0	0	0	1	1	1	1	1	1	0	1	m_0
0	0	0	1	0	1	1	0	0	0	0	0	m_1
0	0	1	0	1	1	0	1	1	0	1	0	m_2
0	0	1	1	1	1	1	1	0	0	1	0	m_3
0	1	0	0	0	1	1	0	0	1	1	0	m_4
0	1	0	1	1	0	1	1	0	1	1	0	m_5
0	1	1	0	0	0	1	1	1	1	1	0	m_6
0	1	1	1	1	1	1	0	0	0	0	0	m_7
1	0	0	0	1	1	1	1	1	1	1	1	m_8
1	0	0	1	1	1	1	0	0	1	1	0	m_9
1	0	1	0	1	1	1	0	1	1	1	0	m_{10}
1	0	1	1	0	0	1	1	1	1	1	0	m_{11}
1	1	0	0	1	0	0	1	1	1	0		m_{12}
1	1	0	1	0	1	1	1	1	0	1		m_{13}
1	1	1	0	1	0	0	1	1	1	1		m_{14}
1	1	1	1	1	0	0	0	1	1	1		m_{15}

Sop LED EQN.

$$\overset{m_2}{}\qquad\qquad\overset{+m_6}{}$$

$A = m_0 + m_3 ~~+ m_4~~ + m_5 + m_7 + m_8 + m_9 + m_{10} + m_{12} + m_{14} + m_{15}$

$B = m_0 + m_1 + m_2 + m_3 + m_4 + m_7 + m_8 + m_9 + m_{10} + m_{13}$

$C = m_0 + m_1 + m_3 + m_4 + m_5 + m_6 + m_7 + m_8 + m_9 + m_{10} + m_{11} + m_{13}$

$D = m_0 + m_2 + m_3 + m_5 + m_6 + m_8 + m_{11} + m_{12} + m_{13} + m_{14}$

$E = m_0 + m_2 + m_6 + m_8 + m_{10} + m_{11} + m_{12} + m_{13} + m_{14} + m_{15}$

$F = m_0 + m_4 + m_5 + m_6 + m_8 + m_9 + m_{10} + m_{11} + m_{12} + m_{14} + m_{15}$

$G = m_2 + m_3 + m_4 + m_5 + m_6 + m_8 + m_9 + m_{10} + m_{11} + m_{13} + m_{14} + m_{15}$

Lab 2: 2844
1-15-19 12:45pm
[illegible]